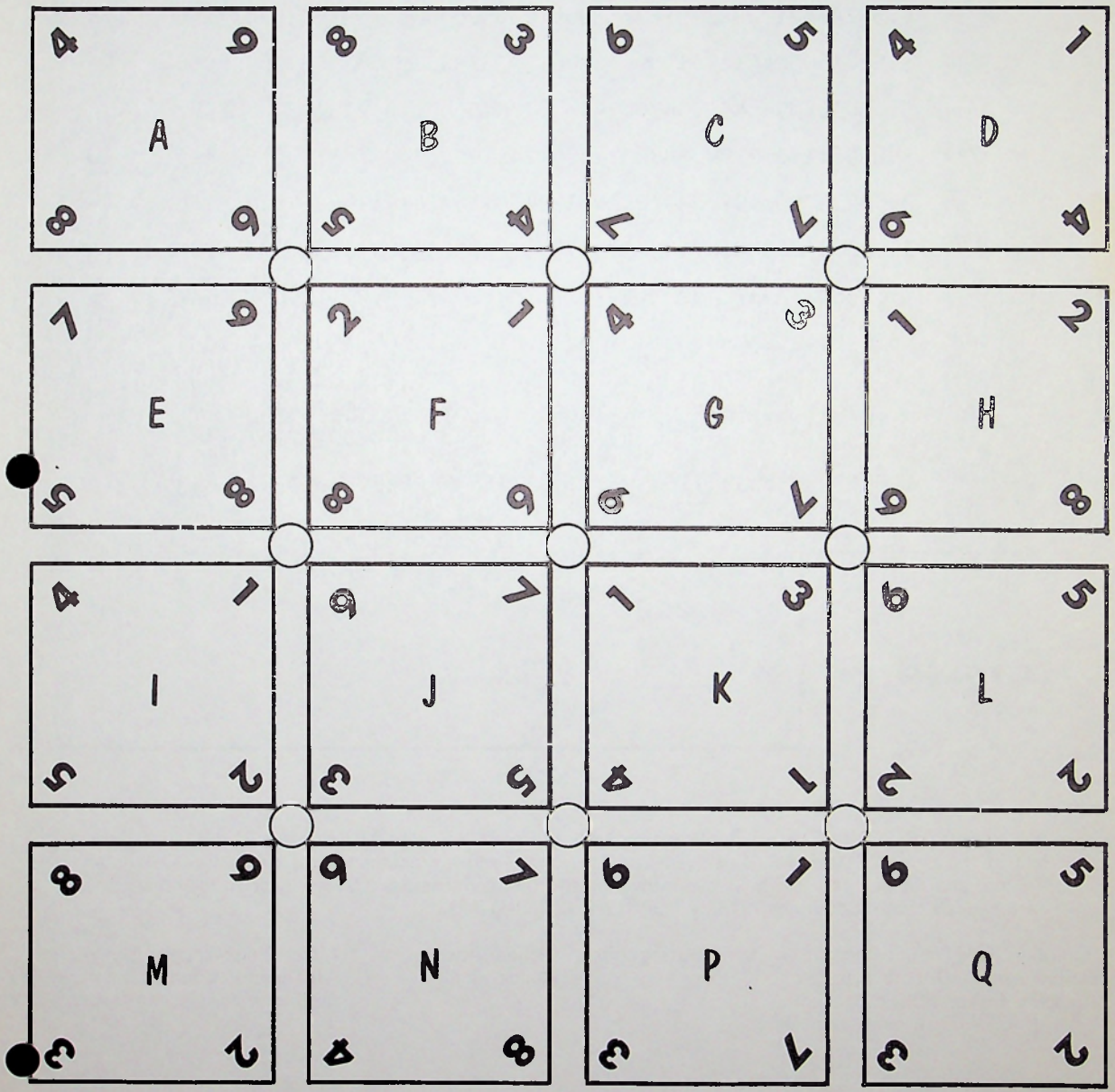
Popular Computing 34

January 1976 Volume 4 Number 1



A Problem for Backtracking

Log 34	1.531478917042255123753908789052830056775757259887155
ln 34	3.526360524616161389666766739331303103663703146946000
$\sqrt{34}$	5.830951894845300470874152877545583076521398334885972
$\sqrt[3]{34}$	3.239611801277483384071469924272029700378378968526529
$\sqrt[5]{34}$	2.024397458499885042510817245541937419114621701073118
$\sqrt[10]{34}$	1.422813219821872839528117387131165958496133027248787
$\sqrt[100]{34}$	1.035892739572482224397017278481013794750977706439141
e^{34}	583461742527454.881402902734610391019003659238941108105782942120431667674211950581147103866
π^{34}	80001047150456339.552949251914152948880188166798408736069745245731536364949009972672316 6
$\tan^{-1}$ 34	1.541393038590891503957218827416327284843325036165643

PRINTED BY
PRINTING & STATIONERY
argold
17002 Ventura Blvd., Encino, Ca. 91316

POPULAR COMPUTING is published monthly at Box 272, Calabasas, California 91302. Subscription rate in the United States is $18 per year, or $15 if remittance accompanies the order. For Canada and Mexico, add $4 per year to the above rates. For all other countries, add $6 per year to the above rates. Back issues $2 each. Copyright 1976 by POPULAR COMPUTING.

Publisher: Fred Gruenberger
Editor: Audrey Gruenberger
Associate Editors: David Babcock
Irwin Greenwald

Contributing editors: Richard Andree
William C. McGee

Advertising Manager: Ken W. Sims
Art Director: John G. Scott
Business Manager: Ben Moore

Backtracking II

BY THOMAS R PARKIN

It is suggested that the serious student of this backtracking topic provide himself with a set of cut-out squares corresponding to the markings on the problem on the cover, in order to aid in following the discussion.

Last month we saw that backtracking is a simple procedure, in concept, and often quite difficult to apply, largely because of the two requirements to generate the solution space of the problem systematically and to be able to detect the undesirability of further exploration along some branch at an early point. We shall apply backtracking to the problem on the cover as an illustration of how it works and where the difficulties lie in its application.

The problem consists of an arrangement of 16 squares, each capable of being placed in any one of four rotational orientations in any one of 16 locations in the larger square. We thus have 16! ways to locate the 16 small squares in the larger square, and 4 to the 16th power ways the individual squares can appear due to rotations for each way they are located. This gives us

$$4^{16} \cdot 16! \text{ or about } 3.5 \cdot 10^{18}$$

possible arrangements comprising the solution space. The end case (or cases) of interest must satisfy the condition that at each of the interior points of the larger square, where 4 small squares meet, such points must be surrounded by 4 identical numbers on the corners of the small squares. Now, even if we could construct the individual cases and test them in one microsecond each, it would still take about 100,000 years to try every case! So, let us apply some analysis and backtracking to see if we can cut that time down.

We shall leave internal coding details until later and concentrate on how we should go about exploring the solution space in a way that will allow us to throw out very large numbers of cases without considering them.

First, let us note that there are seven of each corner number from 1 to 9 except for the number 4, of which there are 8 each. Next we note that there are 9 places where four corners meet, so we either have all the numbers at one each of the corners or all but one at seven corners and the number 4 at two corners.

Now, we could start by picking any one of the 16 squares, rotate it to any one of its 4 positions, and place it down in the, say, upper left corner. Next we would search among our remaining 15 squares for one (or more) which would match that square at the corresponding corners (lower right for the first square; lower left for the second square). We would find one or more such matching squares at this stage of the process, of course, but maybe we won't be so lucky later. We now continue this process (i.e., in an orderly way), progressing from left to right, top row first and then successive rows, selecting from our remaining store of squares one which matches at the corresponding corners according to the conditions of the problem. Place that square in position and continue. Thus we have our systematic generation mechanism.

Sooner or later we shall either come to a halt because there is no remaining square which can be selected to match in the next place where we have a square to go (this is our test mechanism) or we shall use up all 16 squares and we have a solution to the problem.

Let us look at this process in more detail. Suppose we simply start with square A where it is given in the diagram and look for a square to match it on the right. We can now pick from C, F, H, L, P, or Q, each of which has a 6 to match A's 6 in the lower right corner. Let's just take this next square in alphabetical order (which is the rule we will follow all the time), or square C. The process continues this way, then, until we have A, C, E, B across the top row, with C and E each rotated 90° counterclockwise. F now goes in the first position of the second row. At this point, the test for a matching square gets somewhat harder--there are three corners to match simultaneously: the 6's corner, the 7's corner, and the 8 corner (on F). We have the situation shown in Figure 1.

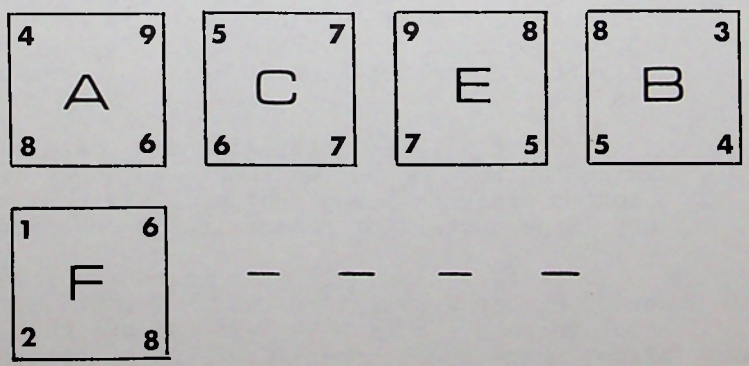


Figure 1

We start looking through the remaining squares for one which will match at the 8, 6, and 7 corners. The only candidates (with at least the 6) are H, L, P, and Q. H comes close, since at least both the 8 and the 6 match, but not the 7 corner; there, H has a 1, and hence won't do. None of L, P, or Q will work either. Well, here we are at a road block. We still have 11 squares to use (representing 11!·4 to the 11th power possible cases) but we know that not one of these can possibly be a solution if the first 5 squares we have placed in the first 5 locations as shown remain exactly in that arrangement. We have thus eliminated a very large part of our solution space simply by picking five squares in some logical systematic order (in this case, alphabetically), in the order of potentially eligible squares for each next location.

Now we backtrack. Let us undo the last step we took prior to reaching the impasse; that is, backtrack one level, discard the choice we made (F), and choose the next eligible square for that location and try again to continue. This results in trying H in the first location of the second row. Again we are blocked, because among F, L, P, and Q none fits properly to match at the 3 corners where it must match. Again, we backtrack one level and try L. Again we are blocked. We continue this process of backtracking, proceeding until blocked, and repeating the process until we have exhausted all the squares that will possibly fit in this fifth spot. Of course, if one fits and we can put in still another that fits, we simply continue the process until we either have a solution or are again blocked, whereupon we repeat the back up-try again cycle. In this case, we shall find that none of F, H, L, P, or Q can be used as the first square of the second row (always holding the first row constant as we have selected it) and allow us to pick any square to fit in the second spot in the second row.

We are now in the position of having exhausted all possibilities for the 5th square, so let's backtrack again to the 4th square.

At this point we have now proven (note that this process constitutes an actual proof) that with the first four squares fixed as shown in Figure 1, there is no possible solution which satisfies all the conditions of the original problem. Let us now try the next eligible square for the 4th position. Possibilities are I, J, L, and Q and, of course, we pick I, since we always select from among those eligible in alphabetic order. Then, we go to the 5th position where the candidates are, again (but not necessarily) F, H, L, P, and Q. At this point we have a choice--either continue the process exactly as we have been doing (in which case we will find that no combination will allow us to pick both a 5th and 6th square) or we can get just a bit clever and, having remembered what we did, simply reject the choice of I

for the 4th place and backtrack again immediately. In order to take this latter step, we have introduced the complication in our program of storing previous sequences of actions. In a simple case like this one, perhaps that refinement would not be justified, but the idea comes in the category of tricks to shorten running time which could be very valuable in some more extensive problem.

In this sample problem, we will discover that we run through all of the possible 4th squares without ever being able to get a 6th one in place. So, again we backtrack to the 3rd square, and repeat the continuing constructive process. (At this point it is no fair indicating what will happen--either try it by hand or write a computer program to solve the problem.) If, perchance, we continue this going forward and backtracking process until we get to the point where we must change the first square, we must now remember that we simply placed it down in the orientation it happens to have in the original statement of the problem on the cover. Hence, we must rotate that square by 90°, and proceed as before. If we should find ourselves back at level #1 again, again we must rotate, and, of course, still again, until we have tried the first square picked in all its possible orientations. Indeed, if at some point we have found more than one matching number on, say, the 3rd (or any other) square, we must remember to try the second possible orientation of that square before we discard it in that spot.

However, suppose we have exhausted square A in all its four orientations in the first place, then we must start the process anew with B and so on. If we follow this procedure through all 16 squares, each in all four orientations, we will have totally exhausted the solution space of the original problem without having actually looked at more than an extremely small fraction of the possible end cases. Furthermore, in the process we will have found <u>all</u> of the solutions (if any) to the original problem. We could also have stopped after finding the first solution (if any) if we wished. Frequently in practical combinatorial problems, all, or at least the number of, solutions are wanted, and if there is none, running through all cases is necessary to demonstrate that there is no solution. The application of backtracking through the use of suitable tests to a suitable orderly generation of cases can very materially shorten the solution time for a great many problems. Indeed, even in this simple case, a program running a few seconds to a few minutes can solve this problem which would be unthinkable to do by brute force.

While it might not be necessary, for this particular problem, to be terribly clever about the lexicographical order of generating the solution space, perhaps a further point could be made to illustrate that this process is subject to refinement and different approaches, which can be powerful aids in shortening the running time for some problems.

For example, it is obvious that the requirements of this problem place the most restrictions on the choices of the four center squares. These must not only match in the center, but at two other places for each square. Thus, if one were to start the solution by picking one of the four center squares and then trying to fit 3 more around that center point, it turns out to be much tougher to get even the 3rd or 4th square into the pattern, and the blockage at any point excludes even more cases at each application of backtracking. However, the coding and programming become quite a bit more difficult, and perhaps too hard to implement for their advantageous use.

We are now in a position to produce a flowchart for this problem. Figure 2 is the flowchart at a fairly gross level showing the basic logic we have used. It is at this point that coding details enter the picture. For example, how should we represent the squares for internal handling by our program? How do we remember where we are in each list of potential next squares for each position? To what extent can we trade storage space for running time? Should we maintain lists or recreate them each time? Answers to these and a host of other questions will determine how we design our detailed flowchart and how we program the problem solution. For example, it becomes quite clear shortly that each square can be thought of as a set of 4 triples; that is, the square consists of four corners, each having a digital value, and each corner has a left and a right corner corresponding to it. We will find that we are concerned about the two values on either side of a corner of a square when it is time to match that square into position as the next square in sequence. Hence, perhaps we should adopt an internal code representation of a square which looks something like Figure 3.

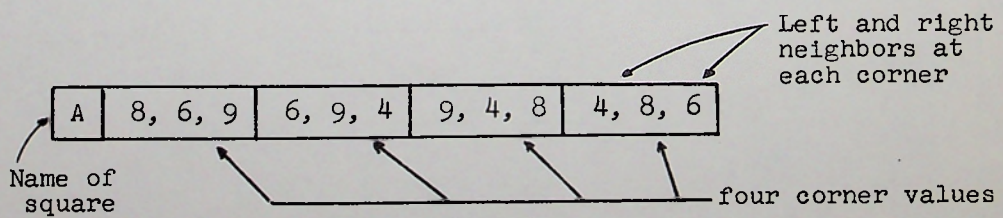


Figure 3

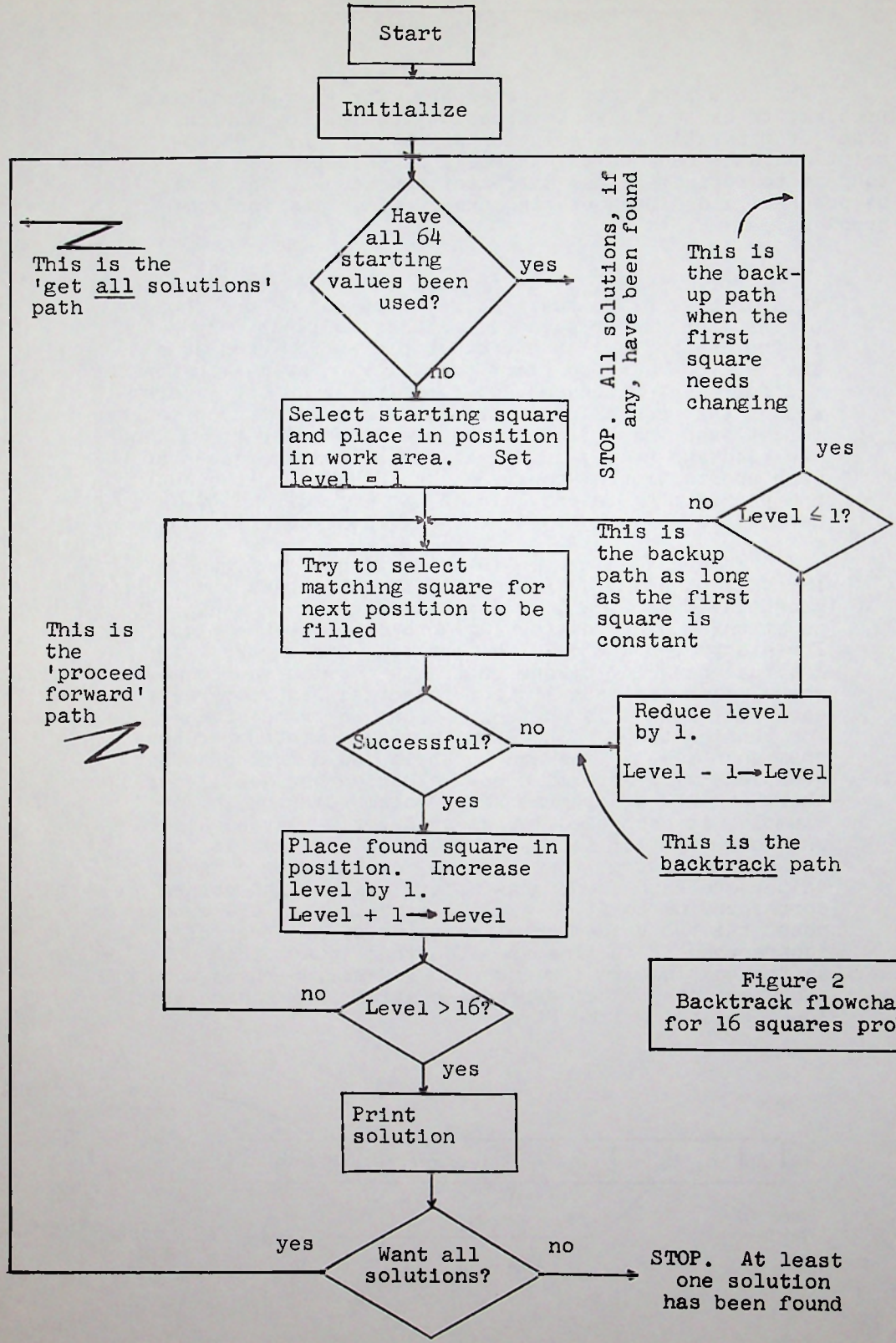


Figure 2
Backtrack flowchart
for 16 squares problem

This may seem like a redundant description of a single square, which can be represented by 5 simple items such as (A, 6, 9, 4, 8), but remember that we must be able to remember the orientation of the square when we put it down and it must match in from 1 to 4 places with other squares. We may find the redundant coding of the squares a bit easier to work with.

Similar questions of coding and indexing of work space and provision for lists and counters and pointers will comprise the preliminary considerations for our program. While these details can materially affect the ease of creating the program, they are merely details. The essential items to be thought through when potentially applying backtracking to a problem are these: 1) Devise an orderly procedure which will generate the totality of end cases in the solution space of the problem, and 2) Devise the most general criteria or tests which can detect dead-ends in the generation process at the earliest possible time. With these two tough (usually) questions answered, the application of the conceptually simple process of backtracking becomes easy. □

The CSR Function

$$A = \sqrt{a_1 + \sqrt{a_2 + \sqrt{a_3 + \sqrt{a_4 + \sqrt{a_5 + \sqrt{a_6 + \sqrt{a_7 + \cdots}}}}}}}$$

The function A, named CSR for "continued square root," is most interesting.

If the successive a's are:

1, 5, 11, 19, 29, 41, 55, 71, 89, 109, 131,...
(successive differences are 4, 6, 8, 10,...)

β

the CSR converges to precisely 2.

But if the a's are the factorials:

1, 2, 6, 24, 120, 720, 5040, 40320, 362880,...
(which are vastly larger)

the CSR converges to the following (computed by Dorothy Cady):

1.82701 47176 08592 22637 38431 92852 89247 37479 36296
08254 85442 61624 62956 21001 52387 09672 78310 72441
66143 0554

This is fun. If the a's are all one, the series converges to the Golden Mean, $(1+\sqrt{5})/2$.

Other values to try for the a's are the following:

A) Consecutive integers starting with 1.
B) Consecutive odd integers starting with 1.
C) Consecutive even integers starting with 2.
D) Fibonacci numbers: 1, 1, 2, 3, 5, 8, 13,...
E) Squares, cubes, and higher powers.
F) Prime numbers, starting with 2.

Herman P. Robinson has shown that the a's can be chosen to converge to any positive number, if both negative and positive terms are used, and that there are infinitely many ways in which this can be done for any number desired. To produce the integer A, one method could be:

$$a_1 = 1$$

$$a_n = (A^2 + (n - 2)K - 1)(A^2 + (n - 2)K - 2) - K$$

with K arbitrary

For A = 2, K = 1, we have the sequence given above (at β) that converges to exactly 2. For A = 2, K = 5, the sequence of a's is the following:

1, 1, 51, 151, 301, 501, 751, 1051, 1401, 1801,...

The sequence of a's can be made to converge to irrational (or transcendental) numbers, and even have various constraints applied to its values. For example, the following sequence contains only prime numbers and converges to π:

5	17	37	53	131	181	263
317	859	887	1637	2837	3413	5861
6491	10531	13399	14083	14563	21433	29717 ...

□

PC34-12

Storage Filling

Words of storage from address 0001 up are cleared to zero. Starting at word 0001, each word of storage is to be loaded with a number, according to this scheme:

If the contents of a word is non-zero, leave it alone.

If the contents of a word is zero, it is to be supplied with a number. If its address is A, then link it to the word at address B, where B = 3A+1. Word A is to be supplied with the number B, diminished by the amount in the first word to the left of word A. Word B is to be supplied with the number B, increased by the amount in the first non-zero word to its left.

The contents of storage will appear as follows, for the first few words:

1	2	3	4	5	6	7	8	9	10
4	3	7	8	8	0	15	0	0	25

11	12	13	14	15	16	17	18	19	20
0	0	0	0	0	41	0	0	0	0

At the stage shown, words 1 through 5 (and their associates, words 7, 10, and 16) have been supplied. Word 6 is next, so at this point A = 6 and B = 19. Word 6 is supplied with 19 minus 8, or 11. Word 19 is supplied with 19 + 41 = 60.

We now have some problems:

1. What will be the contents at address K?

2. Given the contents of a word as J, is its address uniquely determined?

3. After the process has passed L words, will any word at an address less than L have zero for its contents?

4. If the maximum absolute value that a word can hold is M, how many words can be filled before the process must terminate due to overflow?

PROBLEM 113

A financial institution advertises:

> Send us $100 per month for 12 years and we'll send you $100 per month forever

This is simply a clever way of expressing confidence that the institution involved can maintain a 6% interest rate (compounded monthly) indefinitely. At that interest rate, $100 per month for 144 months builds up a reserve of over $21000, and $21000 at 6% can then generate $100 per month in interest without disturbing the principal. As a matter of fact, the build-up period could be 139 months, rather than 144, and the same offer could be made (the amount available at the end of 139 months would be $20004.84).

They could also change the offer to read:

> Send us $100 per month for 12 years and we'll send you $200 per month for the following 12 years

Again, the same offer could be made using 139 months instead of 144.

All the above assumes constant interest rates, and the problems could be solved quickly and readily with compound interest tables.

Suppose, however, that the interest rate were not constant. Assume that a fund of exactly $20000 is available for payout, at an initial rate of 6%, compounded monthly, but that the interest rate rises by .00008333333 per month; that is, the rate rises to 7% at the end of ten years and continues to rise at that rate. How much can be paid back per month, so that the fund is completely depleted after 139 months? □

PROBLEM 114

Fibonacci

In issue 25 and again in issue 30, selected pairs of successive terms in the Fibonacci sequence (numbered so that the 12th term is 144) were given in full. The following list gives successive pairs to 50 significant digits, plus the proper exponent.

1500	1.35511256685631019516369368671484083777860107124 18	E313
1501	2.19261819175562414066861037063099159584869623576 78	E313
2000	4.22469633339230487870672560234148278257985284025 07	E417
2001	6.83570225957580664704539654917058010705540802936 55	E417
2500	1.31709051675194962952276308712531641206660696499 25	E522
2501	2.13109722236481725896324299517047974318182053637 01	E522
3000	4.10615886307971260333568378719267105220125108637 37	E626
3001	6.64390460366960072280217847866028384244163512452 78	E626
3500	1.28013529779468136153585136825101961538900481122 07	E731
3501	2.07130242203026275341619953729934146420769196734 46	E731
4000	3.99094734350044227920812480949609126007925709828 20	E835
4001	6.45748844909481735313769490153695956444139006401 51	E835
4500	1.24421697659866000931127091770892036121474434466 30	E940
4501	2.01318535751626443112205493352144341805943611692 51	E940
5000	3.87896845438832563370191630832590531208212771464 62	E1044
5001	6.27630280048895708603525310834968405547852870273 65	E1044
5500	1.20930645965564463719333621498229900647529023099 25	E1149
5501	1.95669895453763648302504287086161222261066233183 52	E1149
6000	3.77013149387799945333900224880775791236117337930 10	E1253
6001	6.10020089915101922526090032265981303750962193094 42	E1253
7000	3.66434830503723283227635896728160492185715439341 76	E1462
7001	5.92904010416831024406728298679136556671772842955 71	E1462
8000	3.56153320046062673976891490542746038714136953911 02	E1671
8001	5.76268177687862267446233299765388917247126267243 13	E1671
9000	3.46160291286684746313289272940653195821004938840 57	E1880
9001	5.60099116857419990464943760131738213410973574450 56	E1880

10000	3.364476487643178326662161200510754331030214846068O	E2089
10001	5.443837311356528133873426099375038013538918455469 6	E2089
11000	3.270075257282751396758178170577984303744503488195 9	E2298
11001	5.291092912053548874786829393673479632920072442981 2	E2298
12000	3.178322757661353936832333682339347214847452511242 8	E2507
12001	5.142634249113365925793965792899545208268344435268 3	E2507
13000	3.089144670102194414086967697539394617774793539721 3	E2716
13001	4.998341072390931668844753276048731615003528726517 3	E2716
14000	3.002468761178461090995494179715025648692747937490 8	E2925
14001	4.858096505746540834669206297472292420409172861639 1	E2925
15000	2.918224824204913830236407223698513202230962655711 8	E3134
15001	4.721786952377237415507769919282445944935758428591 4	E3134
16000	2.836344622371117843130174614769981827933230126692 2	E3343
16001	4.589302002804454039786760417097735818191535963671 0	E3343
17000	2.756761833470257826431545066175073620252768077808 7	E3552
17001	4.460534345443354646935622154307364635442295499989 0	E3552
18000	2.679411996178763337699310136197842664904176516929 8	E3761
18001	4.335379679681442456357828256732543816555586874933 1	E3761
19000	2.604232457843232402237476415310991787104571582970 8	E3970
19001	4.213736631396027706266932931928527577379191728784 0	E3970
20000	2.531162323732361242240155003520607291766356485802 5	E4179
20001	4.095506670842125091974920367593350749928407432658 4	E4179

Further entries in this table are solicited, as follows:

1. Even 1000's.
2. Pairs of successive entries to allow for interpolation.
3. At least 50 significant digits.

PROBLEM 116

A Merging Problem

PROBLEM 47

Problem Solution

Problem 47 (PC14-5) was the following:

Four blocks of storage are addressed at A through A+9; B through B+9; C through C+9; and D through D+29. Blocks A, B, and C contain numbers which are in ascending order within each block; there are no duplicates among these 30 numbers. We want to merge the 30 numbers into block D. The main routine has already verified that the numbers in blocks A, B, and C are as stated. Draw a flowchart of the logic of a subroutine for the merge.

If blocks A, B, and C were contiguous in storage, the problem would be a simple 3-way merge, and a count of the words moved bo block D would control the merge nicely. The three blocks could be moved to become contiguous, of course, but that would be cowardly and moreover would be impractical for longer blocks. The accompanying flowchart shows a scheme (by Edward Ryan) for the merge.

POPULAR COMPUTING's rating scale for calculators appeared in issue No. 10 and was repeated in issue No. 30. The prices on pocket calculators are still dropping, and the capabilities are still increasing. Since our rating scale is a performance/price ratio, the ratings keep climbing. Consider two widely available machines:

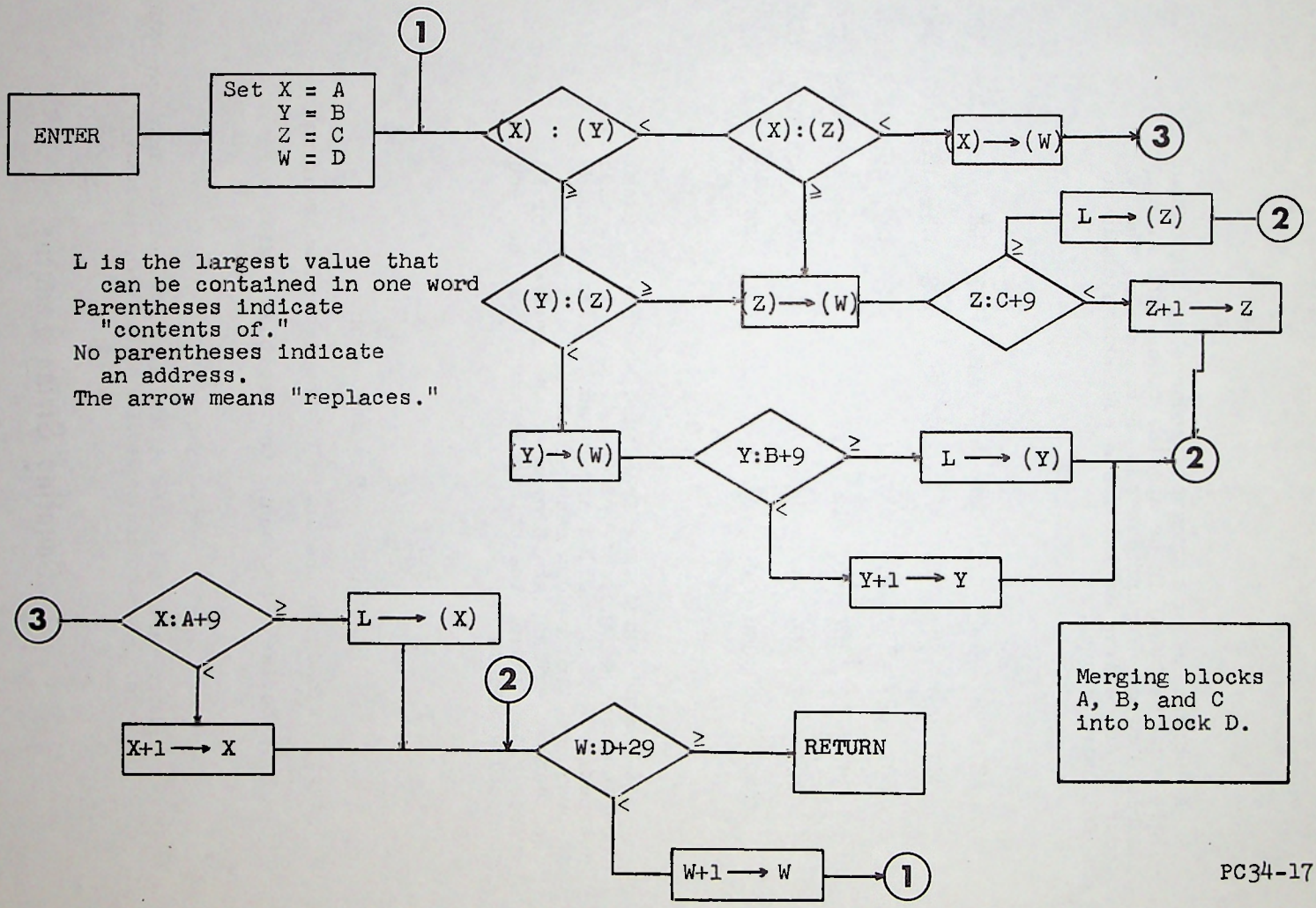
1. The APF Mark 51 (sold by Sears at $35). This machine is an 8-digit floating point calculator for arithmetic operations. When it goes into scientific notation, it uses a 5-digit mantissa and a 2-digit exponent, with a range on the exponent of ± 99. Trig functions, direct and inverse, and natural logs, direct and inverse, all operate to 5 significant digits. There is one word of accumulating storage that functions either + or -. Trig functions can be in degrees or radians. At $35, this machine has a rating on our scale of 32.9.

2. The Texas Instruments' SR-50 is now being discounted as low as $75. It is a fully scientific machine, with all trig and log functions (except common antilog) and every function is calculated to 13 significant digits, with 10 digits in the display plus a 2-digit exponent. Thus, for all pairs of inverse operations (log and exponential; cosine and arccosine; and so on) it is difficult to find any number for which such a pair of functions does not return to the original value. For example, upon entering a number and depressing the reciprocal key repeatedly, the original number appears exactly as entered every other time. At $75, the SR-50 rates 35.0.

On an 8-digit machine, the operation $(4097/4096)^{4096}$ --found by squaring 12 times--approximates the natural log base, yielding 2.7169. With the extra precision of the SR-50, this trick can be extended to

$$(524289/524288)^{524288}$$

---squaring 19 times--to yield 2.7182775. □


ENTER
Set X = A
Y = B
Z = C
W = D
1
(X) : (Y)
(X):(Z)
(X)→(W)
3
L is the largest value that can be contained in one word
Parentheses indicate "contents of."
No parentheses indicate an address.
The arrow means "replaces."
(Y):(Z)
(Z)→(W)
Z:C+9
L→(Z)
2
Z+1→Z
(Y)→(W)
Y:B+9
L→(Y)
Y+1→Y
3
X:A+9
L→(X)
X+1→X
W:D+29
RETURN
W+1→W
Merging blocks A, B, and C into block D.
PC34-17

A subroutine in a program produces as its output 3-digit numbers, (N), in no special order, and with possible strings of duplicates. For example, the output on successive calls of the subroutine could be the following sequence:

387	387	123	006	006	527	527	527	527	129	123	123
387	387	387	387	387	109	109	123	123	123	687	687
687	687	687	687	687	109	109	109	109	153	153	527
387	387	387	687	123	006	527	109	109	109	109	109
109	687	109	496	345	456	006	006	006	006	006	006
006	006	006	109	005	456	567	144	144	238	...	...

A second subroutine is needed which will examine these numbers as they are produced (that is, the second subroutine will be called after each new value of N is produced) and report each new longer string of duplicates. Thus, starting with the sequence of N values given above, the printed output from the second subroutine should be:

387	2
527	4
387	5
687	7
006	9

and so on.

A). Flowchart the logic of the second subroutine.

B). Devise a procedure to test that logic.

PROBLEM 117

[Note: this is Problem 45 (PC13-13) repeated, but in a more explicit form. No solution to Problem 45 has been received.]

Counting String Lengths